DOCUMENT RESUME

ED 307 869 IR 052 754

AUTHOR Teskey, F. N.
TITLE Intelligent Support for Interface Systems.
PUB DATE 88
NOTE 16p.; Paper presented at the International Conference on Research and Development in Information Retrieval (11th, Grenoble, France, June 13-15, 1988).
PUB TYPE Reports - Research/Technical (143) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Artificial Intelligence; *Computer System Design; Information Retrieval; *Information Systems; Information Transfer; *Man Machine Systems; *Menu Driven Software; Online Systems; Pilot Projects; *Programing
IDENTIFIERS *OCLC

ABSTRACT

This paper describes how a language for building information systems that is being developed by the Office for Research at OCLC can be linked to an artificial intelligence environment, Poplog. The demonstration system--ISIS (Intelligent Support for Interface systems)--shows how Poplog could provide some intelligent support for a D interface, which, in turn, could form the basis for intelligent support for an interface system. Two methods in which the link would work have been developed, a request mode and a monitor mode. It is suggested that further research and development would permit the development of the ISIS to the point where it could be used for field trials by OCLC to evaluate different methods of search assistance. (4 figures, 14 references) (EW)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED307869

Intelligent Support for Interface Systems

F. N. Teskey

Information Technology Research Institute
Brighton Polytechnic

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

X This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
F.N. Teskey

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

ABSTRACT

This paper describes how a language for building interfaces to information systems, that is being developed by the Office of Research at OCLC, can be linked to an artificial intelligence environment, Poplog. A demonstration system, showing how Poplog could provide some intelligent support for a D interface, has been developed. It is suggested that this could form the basis for intelligent support for interface systems.

1. INTRODUCTION

In recent years there has been a rapid growth in the number of on-line databases available for searching. One of the claims often made is that these databases provide a vast source of information available 'at the touch of a button' to professional scientists, engineers and others. All too often, however, this source of information is ignored because of the difficulty of accessing it. The growth in on-line databases has led to a profusion of command languages used to access these databases. In practise only professional searchers have the time to learn all, or at least the most important, of these languages. Thus the aim of easy access to vast databases has been hindered by the very growth and diversity of these databases.

We see a need to develop an interface system to help the users and automate as many of the searching procedures as possible [Teskey *et al* 87]. Many of the existing interface systems restrict the user either by providing only a limited menu of functions or by removing much of the interactive nature of the search [Start83 Janke83 O'Leary85 Williams85 Quint86]. Moreover they provide little or no help in developing or modifying the search strategy or adapting to individual user needs. The Advanced Interface Design Project at OCLC has developed a tool for building highly interactive interfaces [Mortland87]. The aim of this paper is to lay the foundations for the addition of some limited intelligence to this interface system, based on the author's recent study visit to OCLC [Teskey87]. There are three main parts to the paper: a review the current state of work at OCLC, a discussion of the functions that would be required to make the existing system more intelligent, and a description of a prototype system.

IR052767

BEST COPY AVAILABLE

2. THE ADVANCED INTERFACE DESIGN PROJECT

The Advanced Interface Design Project at OCLC has developed a series of interfaces to various types of information retrieval systems. The interfaces are implemented in a special purpose language, the D language, developed at OCLC. D is a language for creating interfaces to retrieval systems for bibliographic records, library catalogues and other information systems. The D language is linked to a general purpose retrieval engine, URSA, for carrying out the basic retrieval actions. The primary aim of D is to provide interface designers with a tool for rapid prototyping, and to provide library specialists, who are not necessarily programmers, with a means for modifying easily and quickly as many aspects of the interface as possible.

2.1. Overview of D

The following description is based on an early version of the D language [Mortland87]. The language is still being developed, and new features added. The most important features offered by the D language are its ability to:

(i) present commands and accept responses in a variety of modes including pull down menus, regular menus, command prompts, mouse clicking and movement, or any combination of these,

(ii) create as many screen windows as necessary and define what their contents and modes of interaction will be,

(iii) change the content and the format (placement, fonts, etc.) of displayed data or messages, and

(iv) alter the sequence of states as perceived by the patron (e.g. from the automatic display of a record when only one is retrieved to the display of topics related to the document based on user input or system variables).

The execution of a D program is based on state transitions. Moving from one state to another, say from displaying records to displaying help text or from searching the database to displaying error messages, requires some input either from the user or from the underlying retrieval software. Within each state the value of a variable may be tested and a transfer made based upon the result. The value of the variable may either be accepted from the user or set as the result of a previous action. The D language provides a number of system commands, such as search database, retrieve record etc. which can be included as actions in a state. Much of what occurs in the interface has to do with the display of text in a window and the D language provides a range of facilities for displaying free format and structured text.

2.2. D language concepts

The basic concepts of the D language are the input methods, the state structures and the executable code. These are described in the following sections.

2.2.1. Input methods

Input structures are templates used to specify options for input modes that use the mouse buttons, special character keys, and some parameters for keyboard input. Options include: single or multiple character input, end of input character, cursor display, keyboard input echoing etc. In addition various functions can be linked to function keys and mouse buttons, these include display and selection of choices from a pull pown menu, selecting choices from a scroll-bar, and transfer to specified state labels in the D program. A simple example of an input structure is given below:

```
INPUT A:
specchar=(char=character,label=label_name)
mrd=(function=pull pown,format='title_of_pull pown_menu')
pmi=(format='item1_in_pullpown_menu',label=label_name1)
pmi=(format='item2_in_pullpown_menu',label=label_name2)
end
```

This is an input structure named A. The first line of the structure assigns a D label to a function, or special character key; if this key is pressed control will transfer to the specified D state. The second line indicates the action to be taken on detection of the mouse right button being down (mrd); the template specifies the display of a pull down menu, and the following lines specify pull down menu items (pmi) and the labels to branch to if this item is selected from the menu.

2.2.2. State structures

There are three types of states: active, present and dormant. There is only one active state at any one time and this is the state block the D program is currently executing. The active state can be exited by various methods including accepting an input from the user that satisfies a condition of the state block or moving the mouse arrow into a window that has a present state. There can be any number of present states, they will usually have some representation on the screen and any one of them may become the active state after a successful state transfer. A dormant state is a state block in the D program which has the capability of becoming present in response to a system command. There is one special state, the start state. The start state block is the first block of D code executed in the D program, it does not have to have a condition but must have a branch to label.

2.2.3. Executable code

The executable portion of the D language consists of state blocks and action blocks. A state block is a block of D code where a user's input is accepted into a variable and used to transfer to another state. Each block has an input structure assigned to it and this determines the methods of input that are available to the user. When the user inputs a value then this value is tested in a conditional statement which results in program flow branching to another state. Action blocks consist of several D language statements without any transfers, and operate as subroutines; action blocks may in turn call other action blocks but can not call a state block. Each block may contain any number of D language statements.

A D language statement consists of four separate fields each of which may or may not contain a value. If present, the field must start with an alphabetic character. The fields are as follows:

(i) label
(ii) condition
(iii) action
(iv) branch to

A label acts as an entry point and labels a block; in each block the first, and only the first, statement should have a label. A D language condition places some constraint on the execution of action in the statement it governs. The valid conditional operators are the usual relational operators: equal to, greater than, less than etc. The action part of the statement indicates which particular system command is to be performed. A set of system actions is supplied in the form of procedure calls, usually with parameters. These actions will be either:

(i) commands to the underlying retrieval engine - search, retrieve etc.,
(ii) display commands - show hit list, browse text etc.,
(iii) housekeeping commands - set or clear flags etc. or
(iv) user defined action blocks.

2.3. Demonstration systems

Much of the work described in this paper was based on a full text demonstration system developed by the staff at OCLC. A reconstructed screen dump is shown in Fig.1 below.

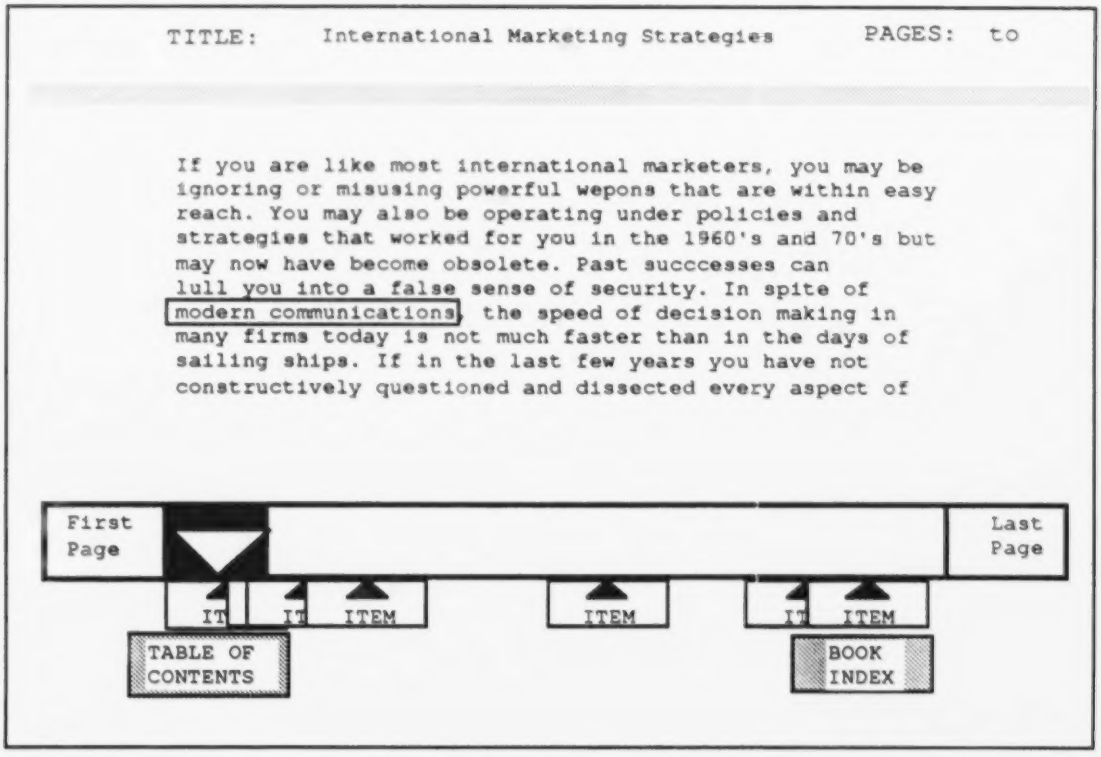
TITLE: International Marketing Strategies PAGES: to
If you are like most international marketers, you may be ignoring or misusing powerful wepons that are within easy reach. You may also be operating under policies and strategies that worked for you in the 1960's and 70's but may now have become obsolete. Past succcesses can lull you into a false sense of security. In spite of modern communications, the speed of decision making in many firms today is not much faster than in the days of sailing ships. If in the last few years you have not constructively questioned and dissected every aspect of
First Page
Last Page
IT
IT
ITEM
ITEM
IT
ITEM
TABLE OF CONTENTS
BOOK INDEX

Figure 1 The full text demonstration system

The data base in this system consists of the full text of a book, including the table of contents and back of book index. The user can browse through the book page by page or use a scroll bar to go directly to selected portions of the book. The user can enter search terms either by selecting words from the displayed text or by direct key board entry. Here the user has selected the words 'modern communication' from the displayed text and selected the find option from a menu; the system has found all occurrences of these selected words and displayed their positions by a series of item markers on the scroll bar.

Another demonstration system has been developed to study interfaces for use in on-line public access catalogues (OPAC's). It is instructive to compare the different styles of interface that can be constructed using the D language. The OPAC system, unlike the full text system, makes very little use of the mouse, and while the full text system is command driven, the OPAC system is menu driven.

3. PROPOSED EXTENSIONS TO THE INTERFACE

The original aim of the D language was to provide a tool for interface designers to develop prototype interfaces quickly and easily. In its present state the D language does not allow these interfaces to adapt to changes in the users' performance or the search results. We wish to study ways in which the language can be extended to allow for more intelligent modifications to the interface that are being built.

Various approaches to the use of AI techniques in information retrieval were discussed in a recent issue of Information Processing and Management. In the introduction to this special issue the editor [Croft87] outlined various areas where AI techniques could be used. These were to develop request models, user models, multiple search strategies and to make flexible use of system facilities. The present system concentrates on the latter two areas. Croft also listed a number of open problems in this area including the formulation of the knowledge used by search intermediaries and the design of efficient access structures. Both these problems are central to this work. Of the existing works presented in this issue of I.P.M., the one that is most similar to our work is the IOTA system [Chiaramella87]. The main difference between our system and IOTA, is that IOTA uses its own information retrieval system driven by an expert system; we believe that this approach is inflexible and have designed a system where the information retrieval system and the expert system operate as co-routines with joint control; this should give greater flexibility in the use of system facilities, including the use of external information systems, and also allow a more natural framework in which to represent the search intermediary's expertise.

3.1 Prototype design

The design of the system is that while the D interpreter is running the interface a second process is running in the background monitoring the conduct of the search and providing assistance when necessary. The assistant can operate in two modes request and monitor. In request mode, if the D program detects a situation it can not deal with, then it requests help

from the assistant to resolve the situation. In monitor mode the assistant is constantly watching the progress of the main D program and if it detects any irregularities it will interrupt the program and attempt to correct the problem.

Within the area of interfaces to information retrieval systems, there is a need for both modes. Subsequent sections of the paper describe how thee modes have been implemented in a prototype system. A few illustrative examples are given of the use of these functions, these form the basis for the demonstration system.

3.2 Prototype functions

One example of where an intelligent assistant could be used in request mode is in dealing with the case where either too many, or too few hits have been retrieved by a search. In this case the interface can easily detect the problem and request help from the assistant. When a search fails the D program should pass the search terms and the hit list to the assistant for further processing; the assistant could then try various modifications to the search and return either a modified search and hit list or a message to say that it too had failed. The interface would then resume its normal processing or switch control to some error display state. At this stage we are concentrating on how the modifications could be made rather than what the modifications might be; they could include changes to the search logic or use of ranking as proposed by [Mitev85] and [Robertson87].

An example of the use of monitor mode is to deal with the case where a user repeatedly issues the same, or a very similar, command and gets an unsatisfactory response. This often indicates that the user does not fully understand the use of that particular command. In this case, by monitoring the actions of the interface, the assistant should identify sequences of similar commands and attempt to discover whether or not these indicate that the user has some problem with that particular command. This requires that the interface provides the assistant with details of the commands that the user has entered. These can then be analysed by the assistant and any possible problems identified; the assistant must then be able to interrupt the main interface and switch it into a suitable tutorial mode to explain the problem to the user.

3.3 Additional functions

Following discussions with some of the staff at OCLC a number of other applications for the search assistant were considered. These included:

(i) error handling - error messages from any of the underlying systems (the search engine, the operating system etc.) should be passed to the search assistant rather than output directly to the user. The search assistant should attempt to discover the severity of the error and take appropriate action. This could range from ignoring insignificant messages to performing a system shut down in the case of a fatal error.

(ii) query construction and modification - if the user does not get any hits for a series of searches then the assistant should be able to suggest alternative search strategies. It is recognised that the construction of search strategies is a major task and may be possible in only a limited number of cases.

(iii) user adaptable interface - the assistant should be able to monitor the way a user is using the system and adapt the interface accordingly. For example, in a command driven mode if the user is repeatedly accessing the HELP facility then it may be appropriate to switch from the command interface to a menu driven interface. Note, that from the full text and OPAC demonstration systems we can see that D already has the capability to support both types of interface.

(iv) performance analysis - the interface can generate statistical data about the use of various commands and functions. This data could be turned over to the assistant to identify areas where the performance of the system could be improved.

The requirements for each of these functions needs to be considered. We believe, however, that the architecture of the prototype system is sufficiently rich and flexible to allow these additional functions to be included with little or no change in the underlying design.

4. THE ISIS SYSTEM

To investigate the ways in which the existing D interface can adapt itself and respond more intelligently to the users' needs it was decided to build a simple demonstration system, ISIS (Intelligent Support for Interface Systems). Since the project was an exploratory one there was a need for rapid development of prototypes. Though the D language does permit some numeric and character string processing, it has not been designed for and is not suitable for general purpose computing or artificial intelligence (AI) work. Thus there was a need to choose another implementation language. The Poplog system [Barrett85] was chosen because it is an AI environment that runs on the same hardware (Sun Workstations) as the D language, it can be linked to other systems and allows for the rapid development of prototypes.

Poplog is an AI programming environment which had its origins in the AI group at Edinburgh in the 60's and is now a commercial product market world wide by Systems Designers. At present the system provides three languages Prolog, Lisp and Pop 11, a syntax directed editor, ved, comprehensive on-line teach, help and reference material and a large library of procedures. The system itself is compiled into a low level machine code, VM, which has been loaded onto a number of different machines. Many of the higher level procedures have been written in Pop 11. At its simplest the Pop 11 language is an weakly typed, procedural language with powerful list and string manipulation functions.

The user can extend the syntax and semantics of the language, and since the system uses an incremental compiler/loader these changes can be made dynamically. Thus the language has very powerful capabilities, but for this work we have used only the basic functions of the language.

Building the prototype can be divided into two stages. First, having identified the areas of the interface which would benefit from the proposed type of extension and in particular having identified the type of data and commands to be transferred between the D language and Poplog, it was necessary to establish the feasibility of making these transfers. This involved several feasibility studies to evaluate different methods of transfer. The second stage was to build a prototype system to demonstrate some of the proposed extensions. These stages are described below, followed by a brief discussion of plans for future work on the ISIS system.

4.1. Feasibility studies

The study of the proposed extensions identified the need for two main types of data transfer between the interface system and Poplog. These are:

(i) synchronous transfer of data initiated by D in request mode, and
(ii) asynchronous transfer of data initiated by Poplog in monitor mode.

Since at this stage there is only a limited amount of control information it was decided to include it in the data flow as command codes and decode it by the D and Poplog systems as required. A third data transfer mechanism may be needed, namely from Poplog to the underlying search engine, for use in modifying searches. Prototype systems were developed to study each of the transfer methods; details of these are given below. All the systems were developed round the D full text demonstration program.

4.1.1 Synchronous data transfer

The simplest method of synchronous transfer of data between D and Poplog is via UNIX files. Both languages have system and file access commands, though those in D are less comprehensive than those in Poplog. This first prototype uses two files D_in and D_out for input and output from D. The full text demonstration program has been modified so that when it detects no search terms it writes a message to that effect to the file D_out, rather than warning the user. The program then waits for a message to come back in the file D_in and displays this message to the user. The Poplog system is running in parallel waiting for a message to be written to the file D_out; as soon as this is detected the Poplog system generates a reply and writes it to the file D_in ready for the D program to access. For this feasibility study the Poplog system did no more than generate a simple message 'Sorry I can not help'. We showed that this was a feasible method of synchronous data transfer. The expected delays due to file accessing did not appear to be significant in the context of a single user interactive system, though this may not be the case for a multi-user system. Other enhancements to this methods for communicating were considered

including the use of unix signals and pipes or unix plug and socket communication. These were not implemented since they would have required some modifications to D to trap unix signals or modifications to the system utilities to use plugs and sockets.

4.1.2 Asynchronous data transfer

The development of methods of asynchronous transfer of data between D and Poplog is more difficult. The D interpreter provides a debug flag which can be set to provide a complete log of each user session. Since this log file is created dynamically during the search session it can be examined by the Poplog system to monitor the progress of the search. At present the log file produces an overwhelming amount of data, so for this second prototype a unix filter was used to remove all information, except that relating to search requests, before writing the information to the file D_log. Once again the Poplog system is running in the background monitoring this file, waiting for new records to be added to the file. If the Poplog system detects an abnormality in the search and it can correct it, then it sets up some corrective information in the file D_cmd. The D interface periodically checks this file and invokes any commands that Poplog has placed there. Once again we showed that this was a feasible method for asynchronous transfer of data between D and Poplog, though in this case there were more serious concerns. First, the debug output from D is not designed for this task and it was only by a happy coincidence that we were able to pick up easily the information relating to searches. Second, the delays due to file access were now more noticeable since the D system had to continually check the file; this was done on each input cycle of the interface and led to a short delay on every command.

4.1.3 Internal data transfer

A third prototype was developed to study data transfer mechanisms between Poplog and the underlying search engine, URSA. The method adopted was to incorporate the existing text searching routines of URSA into the Poplog system. The Poplog system provides a method for loading and calling externally compiled routines. The system supports the transfer of parameters between Poplog and the external routines; at present numeric and character string parameters can be passed out from Poplog and a single numeric result can be returned. Character strings parameters, however, are passed by reference, rather than by value, and so can be used to return values to Poplog. The transfer of parameters was handled by two small routines, a Pop 11 procedure to set up the search parameters and to pick up the resultant hit list, and a C function to pass the parameters received from Pop 11 into the standard retrieve_list routine in URSA and to set up the hit list to be returned to Pop 11. It was also necessary to make minor modifications to the retrieve_list routine to enable it to run in a stand-alone mode. Finally the URSA search routines appear to require a large amount of main memory in which to operate, so it was necessary to reset some of the Poplog parameters to provide enough memory for these external routines. An overview of all the data transfer methods that have been developed is shown in Fig. 2.

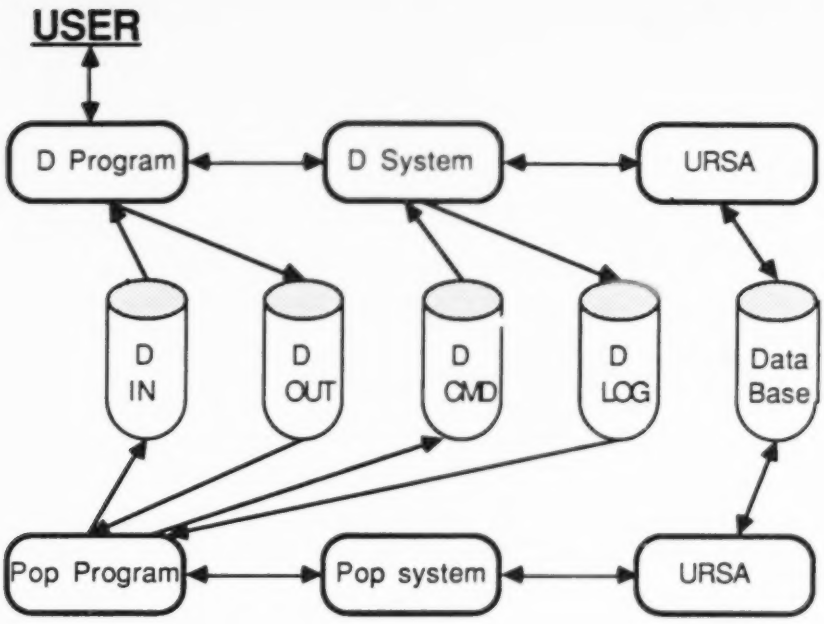

Figure 2 Overview of data transfer methods

The two programmes D and Pop (and their attendant sub-procedures) communicate via the four D_ files and both access the same database files. Within each program the various sub-procedures communicate via procedure calls. Note that the version of URSA in the Poplog system is a cut-down version of the full system. The diagram highlights two points. First, it seems inefficient to have two versions of URSA running at the same time; there is a need to investigate the possibility of having a separate URSA process which communicates directly with both D and Poplog. Second, there is the question of who should read the D_cmd file; since the log file is produced by the D interpreter one could argue that it should also be responsible for processing the D_cmd file that has been generated from the log file. Indeed if the D_cmd file is to contain general D commands, then it will have to be parsed by the D interpreter.

4.2. The first prototype system

A simple prototype system was developed to demonstrate the principles of ISIS. The prototype was based on the existing full text demonstration system implemented in D. Two extensions were added to the system. The first extension was to deal with the case where a single search term retrieved too many hits, the second to deal with one case where the user repeatedly gets only a small number of hits.

4.2.1 Prototype request mode

The existing system deals with multiple hits in a rather unsatisfactory way. If a search returned more than 30 hits then, since the system could not present them all to the user it presented none and invited the user to enter another search. This has now been modified so that the hit list is passed to the Poplog assistant for further processing. Since the prototype is based on the full text of a book, a table of contents and back of book index are available. The assistant first looks for

occurrences of the search term here and if found presents these to the user. For example, the term 'international' retrieves more than 30 hits but it has entries in the back of book index such as 'international competition' and 'international marketing', etc.; these are presented to the user to help refine his search. If the search term does not appear in the table of contents or back of book index then the Poplog system will attempt to locate those areas of the book that are most relevant to the search term by combining the individual hits into ranges of relevant paragraphs and presenting these to the user. If this still results in too much information to present to the user, then a random selection will be presented. Whilst this is far from ideal, it illustrates some of the possible ways of dealing with multiple hits by requesting help from an assistant.

The syntax of the information passed from D in the file D_out is

```
action;item_list
```

In this case the action is 'check_hit_list' and the item_list is a ';' separated list of hits, as produced by D. The Poplog part of ISIS contains a procedure 'check_hit_list' which takes as input a (Pop 11) list of hits and returns a new hit list. This new list is converted to D format, written to the file D_in and read in by the D part of ISIS. In the final system the interface designer will be able to write a Pop 11 procedure of the form

```
action(input_list) -> output_list
```

and access it from D via a statement of the form

```
assist(action,input_list,output_list)
```

4.2.2 Prototype monitor mode

In any interface the user may repeatedly enter commands that are not performing in the way he expects and so does not get the optimal response from the system. For example in the full text demonstration system the user can select terms for searching from the displayed text and these terms are implicitly 'ANDed' together. If a user is repeatedly searching for multiple search terms and retrieving only a few hits, or possibly no hits at all, then that may indicate that he does not realise the difference between 'AND' and 'OR' searching and may benefit from some help on the subject. The Poplog assistant constantly monitors the performance of the interface and keeps track of the number of searches of each particular type. When the frequency of use of any poor search strategy exceeds a certain threshold then the assistant interrupts the D interface to request the display of some suitable help information. At present this consists of some standard help files incorporated into the interface, but there is no reason why it should not include some specific help from the Poplog assistant.

The implementation of the monitor mode of ISIS centres on a Pop-11 database of records of actions of the D interface. The D_log file will contain details of D actions, parameters and (possibly) the current active state; the exact syntax of this is still under discussion. This information will be stored in the Pop 11 database as a series of lists as shown over.

```
[action state
  [1st parameter list] [2nd parameter list] ... ]
```

At present ISIS is looking only for search records so the database consists of records of the form:

```
[search [list of search terms] [hit list] ]
```

The Pop 11 matcher provides a powerful facility for looking for patterns in this database. The pattern template:

```
[search ?terms:multiple ?hits:low]
```

is used to match all search records for multiple search terms which retrieved only a low number of hits. (*multiple* and *low* refer to Pop 11 procedures in ISIS.) The interface designer can set up a number of these pattern templates in ISIS and associate a D command with each one. The D command will be executed whenever the frequency of occurrence of the pattern exceeds a set threshold. The D commands are written to the D_cmd file in the form:

```
action;parameter_list
```

4.4 Worked example

It is difficult to present the results of the prototype system in a written paper since they rely very much on the interactive nature of the interface. The following section is a brief attempt to illustrate the working of the prototype.

The results of the request mode part of ISIS are illustrated in Fig.3. This is the ISIS scroll bar after the same search as was shown earlier in Fig.1. Note that in this case the hit list, represented by the spread of items markers across the scroll bar, has been reduced to three. (In fact in the original system the scroll bar was almost completely covered with ITEM markers making it very difficult to use; for clarity this was not represented in Fig.1.) To indicate that these markers now represent multiple hits the icon has been changed from 'ITEM' to 'MORE'. The important point is not so much the simple clustering that has been achieved, but rather the ability to incorporate such techniques quickly and easily in a prototype system. It is hoped that this will lead to easier testing and evaluation of these techniques.

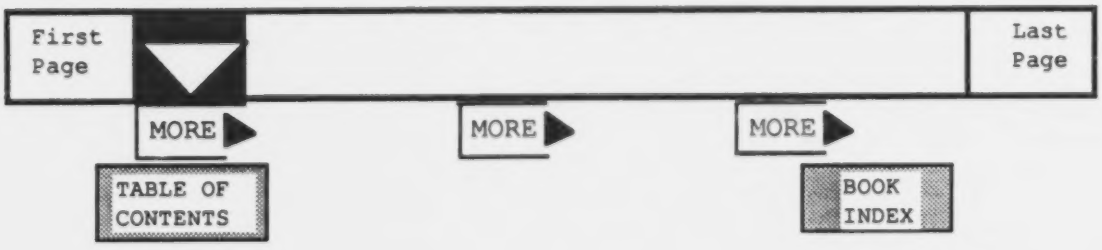

Figure 3 ISIS Scroll Bar

The results of the monitor mode part of ISIS are illustrated in Fig.4. Here the user has just issued a search for 'hot action zones' and retrieved a single item. Prior to this the user had issued a series of similar searches for multiple search terms which retrieved only a few hits. Though this is not a hard error the assistant has detected a possible problem and has requested the D system to display one of the search hints from the help file; this was displayed automatically after the search had been completed.

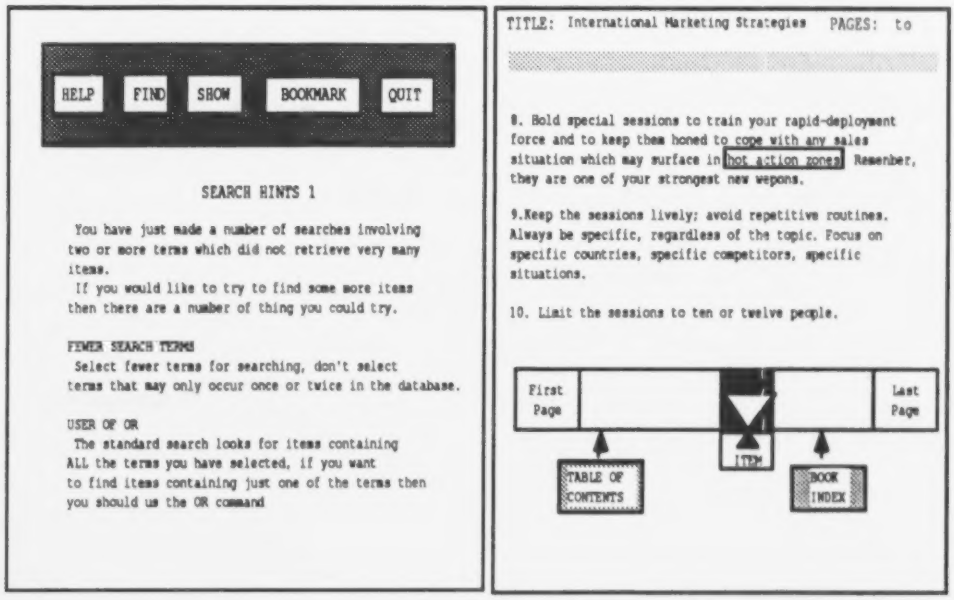
HELP
FIND
SHOW
BOOKMARK
QUIT
SEARCH HINTS 1
You have just made a number of searches involving two or more terms which did not retrieve very many items.
If you would like to try to find some more items then there are a number of thing you could try.
FEWER SEARCH TERMS
Select fewer terms for searching, don't select terms that may only occur once or twice in the database.
USER OF OR
The standard search looks for items containing ALL the terms you have selected, if you want to find items containing just one of the terms then you should us the OR command
TITLE: International Marketing Strategies PAGES: to
8. Bold special sessions to train your rapid-deployment force and to keep them honed to cope with any sales situation which may surface in hot action zones. Remember, they are one of your strongest new wepons.
9.Keep the sessions lively; avoid repetitive routines. Always be specific, regardless of the topic. Focus on specific countries, specific competitors, specific situations.
10. Limit the sessions to ten or twelve people.
First Page
Last Page
TABLE OF CONTENTS
ITEM
BOOK INDEX

Figure 4 ISIS Help Screen

4.4. Future work on ISIS

The demonstration system has shown that an intelligent assistant can be used to extend the functionality of an information retrieval interface. Two possible extensions have been studied and a number of possible methods of implementing these extensions have been investigated. The next stage of the work should be to establish what types of extension are most suitable for each particular application area - full text, OPAC etc. The existing D and Poplog systems, with some of the changes discussed above, should be suitable for this stage of the work. In particular the evaluation of the interface and its extensions will require that the two systems be integrated into an efficient and robust system suitable for field trials. The initial work has identified some specific areas where improvements will need to be made.

The first area for improvement is in inter process communication. The existing method of using files is unnecessarily slow, whilst the use of unix pipes would be considerably faster it may make the system unnecessarily complicated. A better method may be to use interrupts. In many ways this would fit in more closely with the philosophy of states in D - each state as well as defining a structure for user input could also define the action to be taken in the event of receiving an interrupt signal whilst that state was active.

Indeed this method could also be used to provide the interface between D and the URSA search engine. This would provide a clearer interface between the two systems and would mean that each could be used more easily in other applications.

The attempts to incorporate some of the URSA search routines directly in Poplog had to be abandoned because of memory limitations. The URSA routines seemed to be making unreasonable demands on the available Poplog memory space. If ISIS is to have direct access to the search routines then this problem needs to be addressed. However, if as we have just suggested D and URSA are modified to communicate via interrupt handling, then ISIS could make use of the same facility to communicate to URSA.

Another area for improvement is the monitor output from D. At present D can produce either no log output or vast volumes of data in a form more suitable for human reading then computer processing. Whilst it has been possible to use this data in a limited way, a full evaluation of ISIS would require that some of the monitor output from D is designed for use by ISIS.

We have seen the need for parameters in user defined actions. This, combined with some discipline in programming style, should make it much easier to integrate ISIS into existing D interfaces and should also make it much easier to build and maintain those interfaces.

It should be noted that we have not yet introduced detailed user models or request models into the system, but the models developed by [Brajnik *et al* 87] may be used in the system.

5. CONCLUSIONS

We have shown that it is possible to link an intelligent search assistant into an existing interface design language and have developed two main methods in which such a link could work - request mode and monitor mode. We have produced a demonstration system which shows both these methods in action and have identified areas for further improvement. It is hoped that work will continue on this project both at OCLC, Ohio and ITRI, Brighton. This would involve research at Brighton to develop ISIS to a state where it could be used for field trials by OCLC to evaluate different methods of search assistance.

ACKNOWLEDGEMENTS

I am grateful to the British Library Research and Development Department and to the Office of Research at the On-line Computer Library Center for providing funding for the study visit on which this work was developed. n particular I would like to thank Brian Perry and Martin Dillon who initiated this trans-atlantic cooperation. I would also like to thank all the staff at the Office of Research for all their help and support in making my short visit a useful and enjoyable experience.

References

Barrett R., Ramsay A. & Sloman A. (1985), *'POP-11 A Practical Language for Artificial Intelligence'* Ellis Horwood, Chichester.

Brajnik G., Guida G. & Tasso C. (1987), *'User Modelling in Information Retrieval'* Information Processing and Management, Vol.23 No.4 pp.305-320.

Chiaramella Y. & Defode B. (1987), *'A prototype of an intelligent system for information retrieval: IOTA'* Information Processing and Management, Vol.23 No.4 pp.285-303.

Croft W.B. (1987) *'Approaches to Intelligent Information Retrieval'* Information Processing and Management, Vol.23 No.4 pp.244-254.

Janke R.V. (1983) *'BRS/After Dark: the birth of online self-service'* Online Vol.7 no.5 pp.12-29.

Mitev N.N., Venner G.M. & Walker S. (1985), *'Designing an Online Public Access Catalogue - OKAPI, a catalogue on a local area network'*, LIR 39, British Library, Boston Spa.

Mortland B. & Repka A. (1987) *'D: A design language for Retrieval System Interfaces'* submitted for publication by Office of Research, O.C.L.C Ohio.

O'Leary M. (1985), 'EasyNet: doing it all for the end-user', *Online*, vol.9 no.4 pp.106-113

Quint P.(1986), *'Menlo Corporation's Pro-Search: a review of a software search aid'* Online, Vol.10 no.1 pp17-25

Robertson S.E. & Thompson C.L. (1987) *'An operational evaluation of weighting, ranking and relevance feedback via a front-end system'*, British Library R&D Report 5949.

Start C. & Marcinko T. (1983) *'Sci-mate: a menu driven universal online searcher and personal data manager'* Online Vol.7 No.5 pp.112-116.

Teskey F.N., Henry M. & Christopher S. (1987), 'A user interface for multiple retrieval systems' *Online Review* vol.11, no.5 , pp.283-296.

Teskey F.N. (1987), *'Joint report to the British Library and O.C.L.C. on a study visit to develop intelligent support for interface system'* British Library R&D Dept. SI/V/1111

Williams P.W. (1985), *'The design of expert systems for access to information'* Proc. 9th International Online Information Meeting, pp 23-29. Online International, London.